

Střední průmyslová škola strojní   
a elektrotechnická a Vyšší odborná škola, Liberec 1, Masarykova 3

REST API aplikace Rentals

Maturitní práce

Autor **Kateřina Daňková**

Obor **Informační technologie**

Vedoucí práce **Ing. Tomáš Kazda**

Školní rok **2021/2022**

Anotace

Má práce vychází ze původního backendu aplikace Rentals pro správu výpůjček ze školního ateliéru SPŠSE a VOŠ Liberec. Databáze byla nevhodně navržená a při používání se vyskytly problémy, které řeší má práce.

Práce se zabývá vytvořením nové databáze a REST API, také povýšením na framework .NET 6, který vyšel v průběhu vývoje.

Summary

My work is based on the original backend of the Rentals application for managing rentals from the school studio of SPŠSE and VOŠ Liberec. The database was inappropriately designed, and problems occurred during use, which my work solves.

The thesis deals with the creation of a new database and REST API, also upgrading to the .NET 6 framework that came out during development.

Čestné prohlášení

Prohlašuji, že jsem předkládanou maturitní práci vypracoval sám a uvedl jsem veškerou použitou literaturu a bibliografické citace.

V Liberci dne 27.02.2022

Kateřina Daňková

Obsah

[Úvod 4](#_Toc96891010)

[1 Analýza navrženého prostředí 5](#_Toc96891011)

[1.1 Změny 5](#_Toc96891012)

[1.2 Knihovna tus.io 5](#_Toc96891013)

[1.3 PATCH API pro úpravu 6](#_Toc96891014)

[2 Databáze 7](#_Toc96891015)

[2.1 Items 7](#_Toc96891016)

[2.1.1 AccesoryItems 7](#_Toc96891017)

[2.2 Categories 8](#_Toc96891018)

[2.3 Rentings 8](#_Toc96891019)

[2.3.1 RentingItems 8](#_Toc96891020)

[2.3.2 RentingHistoryLog 8](#_Toc96891021)

[2.4 Users 8](#_Toc96891022)

[2.4.1 FavouriteItems, CartItems, InventoryItems 8](#_Toc96891023)

[2.5 Files 8](#_Toc96891024)

[3 Dokumentace API 9](#_Toc96891025)

[3.1 Kontrolér uživatele 9](#_Toc96891026)

[3.1.1 Kontrola uživatele 9](#_Toc96891027)

[3.1.2 Košík uživatele 9](#_Toc96891028)

[3.1.3 Přidání do košíku uživatele 9](#_Toc96891029)

[3.1.4 Odebrání z košíku uživatele 9](#_Toc96891030)

[3.1.5 Vypsání oblíbených předmětů uživatele + filtrování 10](#_Toc96891031)

[3.1.6 Přidání do oblíbených předmětů uživatele 10](#_Toc96891032)

[3.1.7 Odebrání z košíku uživatele 10](#_Toc96891033)

[3.1.8 Vypsání všech uživatelů 10](#_Toc96891034)

[3.1.9 Vypsání předmětů v inventáři uživatele 10](#_Toc96891035)

[3.2 Kontrolér výpůjčky 10](#_Toc96891036)

[3.2.1 Vytvoření nové výpůjčky 10](#_Toc96891037)

[3.2.2 Vrácení předmětů výpůjčky 11](#_Toc96891038)

[3.2.3 Zrušení neuskutečněné výpůjčky 11](#_Toc96891039)

[3.2.4 Vypsání všech výpůjček + filtrování 11](#_Toc96891040)

[3.2.5 Aktivace výpůjčky 11](#_Toc96891041)

[3.2.6 Vypsání dat (od kdy do kdy) všech výpůjček 11](#_Toc96891042)

[3.3 Kontrolér předmětu 12](#_Toc96891043)

[3.3.1 Vytvoření nového předmětu 12](#_Toc96891044)

[3.3.2 Úprava předmětu 12](#_Toc96891045)

[3.3.3 „Smazání“ předmětu 12](#_Toc96891046)

[3.3.4 Vypsání detailu předmětu 12](#_Toc96891047)

[3.3.5 Získání obrázku předmětu 12](#_Toc96891048)

[3.3.6 Vypsání všech předmětů + filtrování 12](#_Toc96891049)

[3.3.7 Vypsání příslušenství předmětu 13](#_Toc96891050)

[3.3.8 Změna příslušenství předmětu 13](#_Toc96891051)

[3.3.9 Vypsání kategorií předmětu 13](#_Toc96891052)

[3.3.10 Změna kategorií předmětu 13](#_Toc96891053)

[3.3.11 Vypsání všech kategorií 13](#_Toc96891054)

[3.3.12 Vytvoření nové kategorie 13](#_Toc96891055)

[3.3.13 Úprava kategorie 14](#_Toc96891056)

[3.3.14 Smazání kategorie 14](#_Toc96891057)

[4 Testování 15](#_Toc96891058)

[4.1 Swagger 15](#_Toc96891059)

[4.1.1 Konfigurace Swaggeru v C# 15](#_Toc96891060)

[4.1.2 Použití Swaggeru 16](#_Toc96891061)

[4.2 Samotné testování 18](#_Toc96891062)

[Závěr 19](#_Toc96891063)

[Seznam zkratek a odborných výrazů 20](#_Toc96891064)

[Seznam obrázků 21](#_Toc96891065)

[Použité zdroje 22](#_Toc96891066)

[A. Seznam přiložených souborů I](#_Toc96891067)

Úvod

Tuto práci jsem si vybrala pravděpodobně proto, že jsem si během druhé poloviny ročníku, kdy jsme byli nuceni zůstat doma, musela zvyknout na to se učit sama. Z webů a programování to většinu času probíhalo pomocí dlouhodobějších projektů, kde jsme se často museli sami vypořádat s řešením. V tu dobu se změnil můj pohled na programování, a od té doby se nebojím věcí, které vidím poprvé. Začali jsme probírat ER modely a framework .NET, a možná právě proto, že to byla první věc, co jsem s tímto přístupem zvládla, jsem se do databází a tvorby API zamilovala. Tudíž jsem si chtěla vyzkoušet tvorbu databáze a API v praxi a získat tak více zkušeností.

# Analýza navrženého prostředí

Za použití frontendu a databáze původních Rentals jsem analyzovala, jak by měla vypadat struktura databáze a endpointy v REST API.

Rentals aplikace byla navržená jako e-shop, což pro výpůjčky ze školního ateliéru, kdy se jeden předmět opakuje maximálně pětkrát, je zbytečně složité řešení. Také předmět v databázi byl rozdělen do dvou tabulek, a to *Typ předmětu* (např. kamera) a poté ve druhé tabulce byla tato kamera na dvou řádcích jako *Kamera\_1* a *Kamera\_2.* V mé práci jsou obě kamery v tabulce předmětu každá jako vlastní záznam s téměř identickými vlastnostmi (popis se může lišit).

Databáze byla odlehčena o tabulku míst, ze kterých se dá půjčovat, jelikož jde o vypůjčovaní věcí pouze z ateliéru naší školy, a naopak se nově přidaly kategorie předmětů, podle kterých se dá v aplikaci lépe vyhledávat. Také byly místa, kde databáze nesplňovala ani první normální formu.

## Změny

Nově se pro správu uživatelů používá školní autorizační server *oauth.pslib.cloud*, díky čemuž se v databázi o uživateli vyskytují pouze potřebné a ověřené informace, které se získávají z oauth.pslib.cloud pomocí Bearer tokenu. Došlo také k přechodu .NET 6 a přidání knihovny tus pro nahrávání obrázků.

## Knihovna tus.io

Tato knihovna se stará o nahrávání obrázků tak, že ho rozdělí na několik menších částí a ty postupně nahrává. Bez použití knihovny se mohlo stát, že když uživatel nahrál příliš velký soubor na méně výkonném zařízení, tak celá aplikace zamrzla a muselo se počkat několik sekund, než se obrázek kompletně nahraje. Další výhodou této knihovny je, že se dá nahrávání pozastavit. Zároveň je jednoduše nakonfigurovatelná jak na backendu a frontendu, kde nám umožňuje i vizuálně skvělou možnost ukázat během nahrávání progres.

Obsah obrázku text

Popis byl vytvořen automaticky

Obrázek 1 Základní konfigurace knihovny tus v .NET 6

## PATCH API pro úpravu

V projektu jsem použila PATCH API pro úpravu záznamů. Jedná se o rozšiřující metodu PATCH, jako je například GET, POST atd. Je to metoda podobná PUT, ten ale mění všechny vlastnosti objektu a v případě kdy bychom chtěli změnit například jméno u osoby, která má jméno a příjmení, příjmení se nastaví na null. Díky metodě PATCH lze upravit pouze jednu (nebo i více) vlastností beze změny ostatních.

PATCH metoda přijímá parametr navíc a to *JsonPatchDocument*. Jedná se v podstatě o kolekci vlastností, které chceme změnit. Pro každou měněnou vlastnost se udává

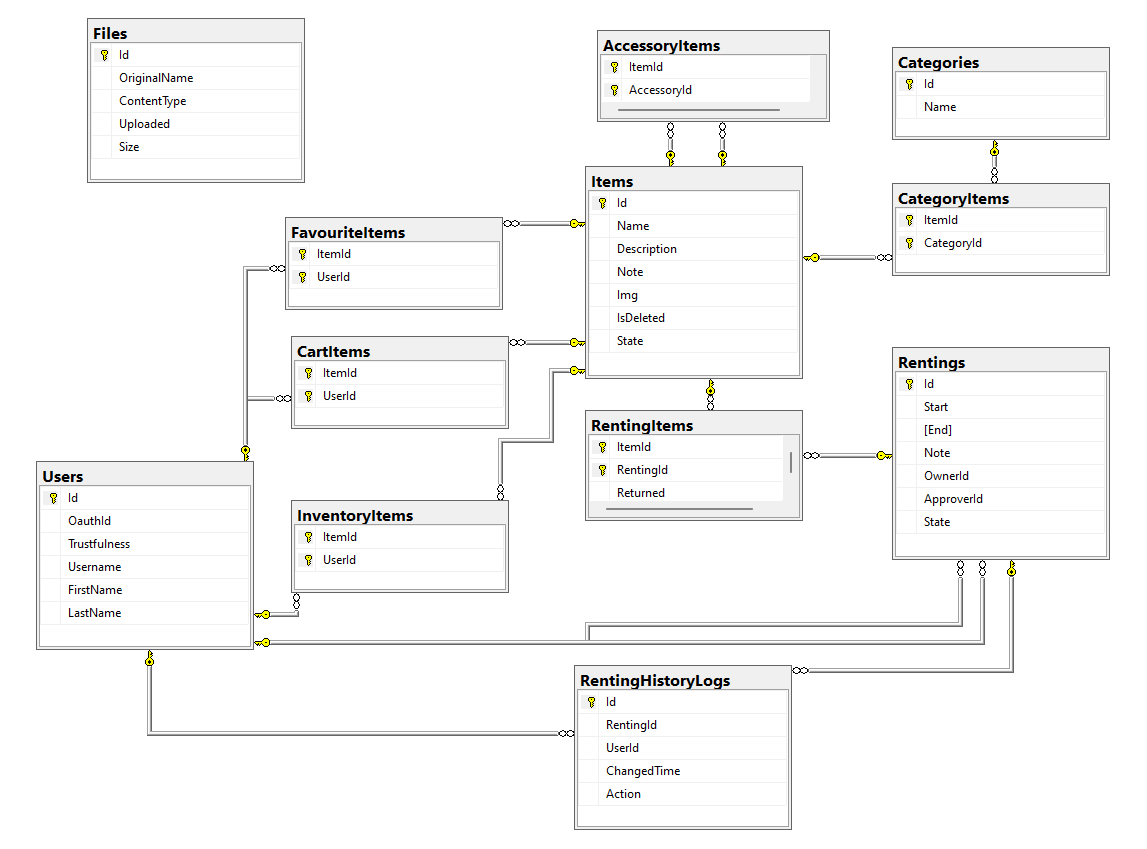
* *path* (hodnota se píše s lomenem – v příkladu měním popis obrázku tudíž *“/Description“)*
* *op* (operace kterou chceme provést, v našem případě nahradit – *replace*)
* *value* (hodnota na kterou se má vlastnost popis přepsat)

Obsah obrázku text

Popis byl vytvořen automaticky

Obrázek 2 Příklad JsonPatchDocument

# Databáze



Obrázek 3 ER model nové databáze

## Items

Tabulka předmětů obsahuje název předmětu, popis, poznámku (např. zda byl poškozen, nebo něco chybí), cestu k obrázku. Dále má výčtový typ stav, který může být buď dostupný, vypůjčený, nebo nedostupný. Jako poslední má předmět vlastnost, zda je smazán. Z bezpečnostních důvodů se data v aplikaci skutečně nemažou a pouze skrývají, aby nedošlo k jejich ztrátě.

### AccesoryItems

Jedná se o vazbu mezi příslušenstvím a předmětem. Jelikož je příslušenství předmět, vzniká zde vazba M:N mezi předmětem a předmětem, protože předmět může být příslušenstvím pro mnoho předmětu a zároveň předmět může mít mnoho příslušenství. Tato vazba by se od verze .NET 5 měla vytvářet automaticky, ale pro přehlednost je vždy vytvářím.

## Categories

Definuje jednoduchou tabulku pro kategorii, která obsahuje jen své jméno.

## Rentings

Výpůjčka se skládá z datumů, kdy by měla začínat a končit a dodatečné poznámky. Má vlastníka a osobu, která výpůjčku schválí a výčtový typ stav, který může být začne, probíhá, ukončená, zrušená.

### RentingItems

Vazba mezi výpůjčkou a předmětem navíc obsahuje, jestli byl už v dané výpůjčce předmět navrácen.

### RentingHistoryLog

Tabulka pro historii úprav předmětu, kam se při každé změně výpůjčky zaznamená, kdo provedl změnu, na jaké výpůjčce, o jakou akci se jednalo a kdy ke změně došlo.

## Users

Tabulka uživatelů vychází z dat oauth.pslib.cz, proto obsahuje jen informace jako je ID z Oauthu, křestní jméno, příjmení a uživatelské jméno, kterým je e-mail. Veškeré ostatní informace se dají získat z tokenu na frontendu aplikace.

### FavouriteItems, CartItems, InventoryItems

Tyto tři vázací tabulky mezi uživatelem a předmětem, které slouží pro oblíbené předměty, košík a inventář uživatele, jsou naprosto stejné. Každý předmět tu má vlastní záznam, kdežto v původní databázi byl například inventář jako textová vlastnost uživatele, což nesplňuje 1. normalizační normu databáze.

## Files

Samostatná tabulka pro nahrané obrázky. Ukládá jejich původní název, typ, datum a velikost

# Dokumentace API

API se skládá ze tří kontrolérů. Kontrolér uživatele, ten se stará o založení záznamu v databázi a umožňuje uživateli například ukládat předměty do košíku, či si je přidat do oblíbených. Kontrolér předmětů obsahuje metody pro různé druhy změn na předmětu + kategorií. Nakonec kontrolér pro administraci výpůjček. Na některých místech se pro přehlednost používá jako parametr objekt, který obsahuje parametry, které potřebujeme. Pro všechny metody je potřeba aby byl uživatel přihlášen.

## Kontrolér uživatele

Kontrolér uživatele přebírá data o uživateli z autorizační aplikace oauth.pslib.cloud, tudíž nepotřebuje metody pro úpravu nebo mazání uživatele.

### Kontrola uživatele

Slouží pro zjištění, zda uživatele máme již v databázi a pokud ne, vytvoří ho. Zavolá se ihned po přihlášení do aplikace. Jako parametr přijímá objekt*,* který obsahuje ID z oauthu, uživatelské jméno, křestní jméno a příjmení.

### Košík uživatele

Slouží pro vypsání předmětů v košíku právě přihlášeného uživatele. Díky atributu *[Authorize]* nejsou potřeba parametry. Vrátí list předmětů.

### Přidání do košíku uživatele

Slouží pro přidání předmětu do košíku právě přihlášeného uživatele. Jako parametr přijímá ID předmětu, který chceme uložit. Vrací metodu pro vypsání košíku uživatele.

### Odebrání z košíku uživatele

Slouží pro odebrání předmětu z košíku právě přihlášeného uživatele. Jako parametr přijímá ID předmětu, který chceme odebrat. Vrací metodu pro vypsání košíku uživatele.

### Vypsání oblíbených předmětů uživatele + filtrování

Slouží pro vypsání předmětů, které má právě přihlášený uživatel uložené v oblíbených. Přijímá nepovinný parametr ID kategorie, pomocí které se vypíší pouze předměty s touto kategorií. Vrací list předmětů.

### Přidání do oblíbených předmětů uživatele

Slouží pro přidání předmětu do oblíbených právě přihlášeného uživatele. Jako parametr přijímá ID předmětu, který chceme přidat. Vrací metodu pro vypsání oblíbených předmětů uživatele.

### Odebrání z košíku uživatele

Slouží pro odebrání předmětu z oblíbených právě přihlášeného uživatele. Jako parametr přijímá ID předmětu, který chceme odebrat. Vrací metodu pro vypsání oblíbených předmětů uživatele.

### Vypsání všech uživatelů

Slouží pro vypsání všech uživatelů, například pro správu výpůjček dané osoby. Na toto má právo pouze zaměstnanec nebo administrátor. Nepřijímá žádné parametry a vrací list uživatelů.

### Vypsání předmětů v inventáři uživatele

Slouží pro vypsání předmětů, které by měl mít uživatel u sebe. Na toto má právo pouze zaměstnanec nebo administrátor. Jako parametr přijímá ID uživatele (oauth id) a vrací list předmětů.

## Kontrolér výpůjčky

### Vytvoření nové výpůjčky

Slouží pro zadání nové výpůjčky, přijímá objekt, který obsahuje kolekci ID předmětů (z košíku), datum, kdy by měla výpůjčka začít a datum kdy by měla výpůjčka skončit a poznámku. Vrací právě vytvořenou výpůjčku.

### Vrácení předmětů výpůjčky

Slouží pro vrácení předmětů z dané výpůjčky, pokud se vrátí všechny předměty, dojde k její ukončení. Na toto má právo pouze zaměstnanec nebo administrátor. Jako parametr přijímá objekt, který obsahuje ID výpůjčky a kolekci ID předmětů, které chceme vrátit. Vrací změněnou výpůjčku.

### Zrušení neuskutečněné výpůjčky

Slouží pro případ, že by si někdo nevyzvedl výpůjčku, a tak bylo potřeba výpůjčku zrušit. Na toto má právo pouze zaměstnanec nebo administrátor. Jako parametr přijímá ID výpůjčky, kterou chceme zrušit. Vrací zrušenou výpůjčku.

### Vypsání všech výpůjček + filtrování

Slouží pro vypsání všech výpůjček. Na toto má právo pouze zaměstnanec nebo administrátor. Má nepovinný parametr stav výpůjčky, díky kterému se dají vypsat pouze výpůjčky s tímto stavem. Vrací list výpůjček.

### Aktivace výpůjčky

Slouží pro aktivaci, respektive fyzické vyzvednutí předmětů výpůjčky, kterou si uživatel vytvořil. Na toto má právo pouze zaměstnanec nebo administrátor. Jako parametr přijímá ID výpůjčky, kterou chceme aktivovat. Vrací aktivovanou výpůjčku.

### Vypsání dat (od kdy do kdy) všech výpůjček

Slouží pro vypsání všech dat výpůjček, které probíhají nebo teprve začnou. Použije se například při zobrazení v kalendáři. Nepřijímá žádný parametr a vrací list objektů, který obsahuje začátek, konec a vlastníka výpůjčky.

## Kontrolér předmětu

### Vytvoření nového předmětu

Slouží pro přidání nového předmětu do databáze. Na toto má právo pouze administrátor. Jako parametr přijímá objekt, který obsahuje název předmětu, popis, poznámku a cestu k obrázku. Vrací právě vytvořený předmět.

### Úprava předmětu

Slouží k úpravě vlastností předmětu. I když na metodu má právo zaměstnanec nebo administrátor, zaměstnanec může měnit pouze poznámku a popis obrázku. Nelze zde měnit ID předmětu a zda je smazaný. Jako parametry přijímá ID předmětu, který chceme měnit, a objekt JsonPatchDocument. Vrací upravený předmět.

### „Smazání“ předmětu

Slouží k nastavení vlastnosti, zda je předmět smazaný. Na toto má právo pouze administrátor. Jako parametry přijímá ID předmětu, který chceme smazat, a objekt JsonPatchDocument (protože měníme jednu vlastnost). Vrací „smazaný“ předmět.

### Vypsání detailu předmětu

Slouží k vypsání vlastností předmětu, pokud není „smazaný“. Jako parametr přijímá ID předmětu a vrací daný předmět.

### Získání obrázku předmětu

Slouží k získání souboru obrázku daného předmětu. Jako parametr přijímá ID předmětu, jehož obrázek se snažíme získat a vrací soubor s obrázkem.

### Vypsání všech předmětů + filtrování

Slouží k vypsání všech předmětů. Má nepovinný parametr ID kategorie, díky kterému se dají vypsat pouze výpůjčky s touto kategorií. Vrací list předmětů. Pro kombinování více kategorií se zavolá endpoint pro každou kategorii.

### Vypsání příslušenství předmětu

Slouží k vypsání všech předmětů, které jsou příslušenstvím k danému předmětu. Použije se například u zobrazení předmětu, aby uživatel věděl, co se k danému předmětu hodí. Jako parametr přijímá ID předmětu, ke kterému chceme vypsat příslušenství a vrací list předmětů.

### Změna příslušenství předmětu

Slouží k přiřazení předmětů jako příslušenství k danému předmětu, původní nevybrané se odeberou. Jako parametr přijímá objekt, který obsahuje ID předmětu, kterému chceme příslušenství přiřadit a kolekci ID předmětů, které mají být příslušenstvím. Vrací upravený předmět.

### Vypsání kategorií předmětu

Slouží k vypsání kategorií, které již předmět má. Využije se pro seznam, na přidání / odebrání kategorie. Jako parametr přijímá ID předmětu, jehož kategorie chceme získat a vrací list kategorií

### Změna kategorií předmětu

Slouží k přiřazení kategorií k danému předmětu, původní se odeberou. Jako parametr přijímá objekt, který obsahuje ID předmětu, kterému chceme kategorie přiřadit a kolekci ID kategorií. Vrací upravený předmět.

### Vypsání všech kategorií

Slouží pro vypsání všech kategorií, použije se například k filtrování. Nepřijímá žádný parametr a vrací list kategorií.

### Vytvoření nové kategorie

Slouží pro vytvoření nové kategorie. Jako parametr přijímá název kategorie a vrací úspěšně vytvořenou kategorii.

### Úprava kategorie

Slouží pro změnu jména kategorie. Jako parametr přijímá ID kategorie, kterou chceme měnit a objekt JsonPatchDocument a vrací upravenou kategorii.

### Smazání kategorie

Slouží pro odstranění kategorie. Jako parametr přijímá ID kategorie a vrací smazanou kategorii.

# Testování

První fáze testování probíhala mnou za pomocí Swaggeru.

## Swagger

Swagger je open source framework pro návrh a tvorbu dokumentace pro RESTful API. Obsahuje nástroje pro automatické vygenerování dokumentace a testování existujícího API, vizualizaci a vyzkoušení navrženého API.

Společnost SmartBear Software se podílela na vývoji OpenAPI a darovala svoji specifikaci swaggeru do nově vzniklé iniciativy. OpenAPI 2.0, která je obsahově shodná s původní specifikací Swagger 2.0, je dnes možné chápat jako specifikaci a Swagger jako nástroj pro implementaci této specifikace. (1)

### Konfigurace Swaggeru v C#

Obsah obrázku text

Popis byl vytvořen automaticky

Obrázek 4 Konfigurace Swaggeru v C#

Na přiloženém obrázku je konfigurace Swaggeru, která se skládá ze tří částí. První, červená část, je základní konfigurace, díky které se generuje Swagger.

V další zelené části je kód pro slovní popis metod, díky které můžeme použít nad každou metodou v kontroléru tag summary, jehož obsah se nám propíše k metodám ve Swaggeru.

Obsah obrázku text

Popis byl vytvořen automaticky

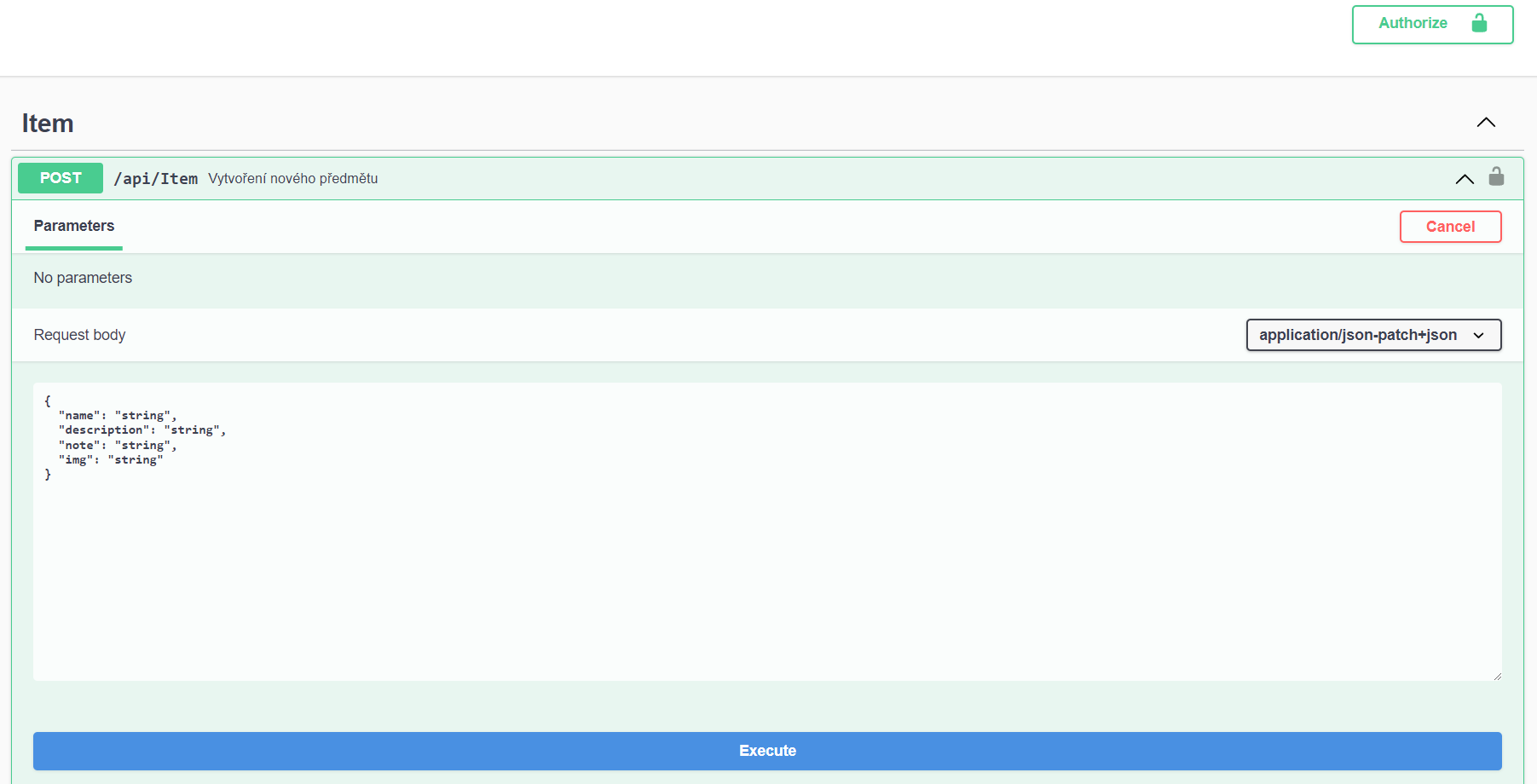
Obrázek 5 Ukázka použití summary

Poslední modrá část zajišťuje možnost testovat zabezpečení – autorizaci API, pomocí JWT Bearer tokenu.

Zatímco se tato konfigurace vyskytovala v programu v builder.Services, je potřeba pro funkční Swagger ještě napsat jeden řádek kódu na závěr Program.cs.

app.UseSwagger();

### Použití Swaggeru

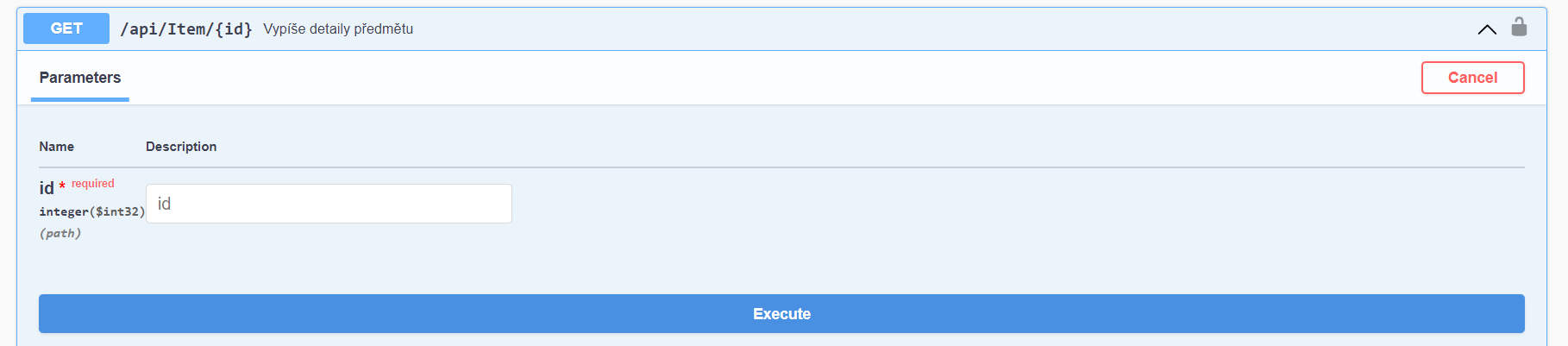


Obrázek 6 Příklad metody ve Swaggeru

V pravém horním rohu se nachází Authorize tlačítko, pro vložení Bearer tokenu, kterým se uživatel bude vůči aplikaci autorizovat.

V levém horním rohu vidíme název kontroléru, a pod ním seznam všech metod, které obsahuje. V tomto případě již máme otevřenou metodu pro vytvoření nového předmětu. Je zde vidět že se jedná o metodu POST (každá metoda má i vlastní barvu), adresu, na kterou endpoint odkazuje, a za tím popis, který jsme předtím napsali k metodě do summary.

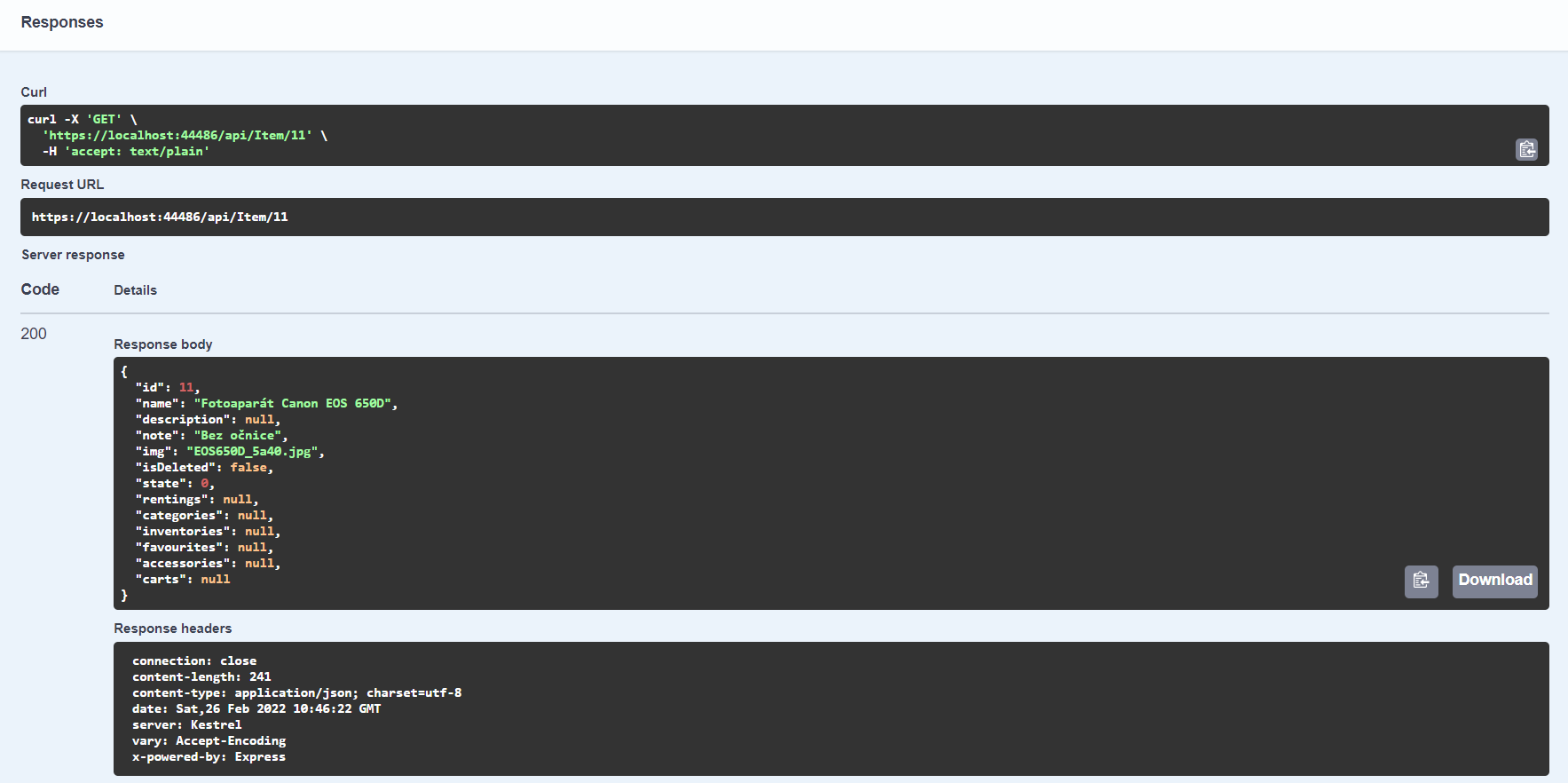
Tato metoda nemá žádné parametry, které by se psali přímo do URL, ale obsahuje tělo požadavku, kam se zapíší vlastnosti, které chceme, aby předmět obsahoval. Vlastnosti se oddělují čárkou, a vlastnosti, které nechceme vyplňovat, nemusíme udávat.



Obrázek 7 Metoda s parametry v URL

Když zkusíme metodu zavolat, v Responses uvidíme Request URL, která reprezentuje adresu, na které bychom metodu zavolali. Dostaneme kód odpovědi (200 pro úspěšné splnění požadavku, 401 pokud uživatel nemá přístup a 404 pokud požadovaná data neexistují).

Odpověď na požadavek nám také vrátí data, v našem případě předmět a hlavičku odpovědi.



Obrázek 8 Odpověď na požadavek

## Samotné testování

Pro každou metodu jsem testovala nejdřív s validními parametry, zda dělá, co se od ní očekává, poté jsem zkoušela i zadat parametry neexistující položky, zda vyhodí kód 404. V případě předmětu dostaneme 404 i pro předmět, který má vlastnost IsDeleted nastavenou na true.

V počátcích vývoje a testování, kdy ještě nebylo API napojeno na autorizační server zde byly metody pro vytvoření uživatele, jeho úpravu a mazání, které ve finální verzi nejsou potřeba, zbyla zde pouze metoda pro přidání uživatele přihlášeného k autorizačnímu serveru do databáze Rentals aplikace.

Pro testování přihlášených uživatelů a jejich práv jsem využívala pomocnou aplikaci napsanou v Reactu, která slouží k vypsání Bearer tokenu a profilu přihlášeného uživatele k oauth.pslib.cz. Tento token jsem pokaždé zadala do Swaggeru a otestovala, zda nastavená práva fungují.

Závěr

Do této maturitní práce jsem šla s cílem vyzkoušet si navržení reálné databáze a API a případně si rozšířit své znalosti. Trochu jsem se obávala, že moje znalosti na databázi v praxi nebudou dostačující, ale nakonec jsem použila převážně věci, co jsem znala ze školy. Některé věci se ale nejdřív použily v mé práci, než jsme je stihli probrat ve škole, čímž jsem v následujících projektech měla výhodu.

Zjistila jsem, jak se nahrávají obrázky do API a jak je poté přes endpoint získat zpět, to pro mě asi byla největší novinka. I když prvotní implementace byla přímo přes nahrávání souboru jako parametr endpointu, pracovat se s tím dalo, kdyby nebyl problém se zamrzáním aplikace. Pak ale přišel pan Kazda, že bude lepší, a i pro obsah endpointu jednodušší použít tus. Příjemná novinka pro mě bylo také použití PATCH API, které velmi zjednodušuje úpravu entit. Do té doby jsem používala pouze PUT, který je přímo v REST API. Ujistila jsem se, jak se píšou metody a k čemu slouží jaký návratový kód.

Autorizace pro mě nebyla úplná novinka, i když jsme ve škole pro správu uživatele použili pouze identitu ASP.NET, na praxi ve firmě mi autorizace za použití Bearer tokenu byla přiblížena, a tak mě nepřekvapilo, když jsem dostala za úkol implementovat autorizaci vůči školnímu serveru. Ovšem claimy a práci s nimi, jsem si vyzkoušela až v tomto projektu úplně poprvé, i když si myslím, že jsme je již ve druháku zmiňovali.

Další věc, co jsem pochytila na praxi ve firmě bylo použití místo několika parametrů endpointu objekt, který všechny potřebné parametry obsahuje. Bez tohoto jsem měla u jen díky parametrům některé metody dlouhé přes dva řádky.

Problémy vyloženě s vývojem jsem neměla, veškeré moje chyby byly naprosto hloupé, i přesto že jsem u jejich řešení strávila i několik dní. Například pozůstatky v databázi po identitě ASP.NET, mi dělaly neplechu při implementaci vlastní Bearer, nebo SPA proxy mělo problém s novou verzí Node.js.

V budoucnu by dala implementovat důvěryhodnost uživatele, která je již v databázi připravená, ale nebyla vymyšlená funkcionalita, proto se v aplikaci zatím nevyskytne. Také by se dali přidat e-maily na upozorňování, že výpůjčka končí, že je předmět dostupný, nebo že uživatel přesáhl termín vrácení.

Seznam zkratek a odborných výrazů

HTML

HyperText Markup Language – značkovací jazyk používaný pro tvorbu webových stránek.

Seznam obrázků

[Obrázek 1 Základní konfigurace knihovny tus v .NET 6 3](#_Toc96890968)

[Obrázek 2 Příklad JsonPatchDocument 3](#_Toc96890969)

[Obrázek 3 ER model nové databáze 4](#_Toc96890970)

[Obrázek 4 Konfigurace Swaggeru v C# 12](#_Toc96890971)

[Obrázek 5 Ukázka použití summary 13](#_Toc96890972)

[Obrázek 6 Příklad metody ve Swaggeru 13](#_Toc96890973)

[Obrázek 7 Metoda s parametry v URL 14](#_Toc96890974)

[Obrázek 8 Odpověď na požadavek 15](#_Toc96890975)

Použité zdroje

1. **Stehlík, Michal.** *Návod k maturitním pracím 2020.* Liberec : Albatros, 2020.

2. **Wikipedia foundation.** Swagger (software). *Wikipedie otevřená encyklopedie.* [Online] 16. Červenec 2021. [Citace: 26. Únor 2022.] https://cs.wikipedia.org/wiki/Swagger\_(software).

1. Seznam přiložených souborů

Na přiloženém datovém nosiči se nacházejí následující soubory a složky:

* **MP2022-Daňková-Kateřina-REST-API-aplikace-Rentals.docx** – editovatelná verze dokumentace maturitní práce
* **MP2022-Daňková-Kateřina-REST-API-aplikace-Rentals.pdf** – tisknutelná verze dokumentace maturitní práce
* **Aplikace** – zdrojové kódy